ESCOLA UNIVERSITARIA POLITECNICA
DE VILANOVA I LA GELTRU

BRANCA: TELECOMUNICACIO

PASCAL. PROGRAMAS

Departament de Matemàtica Aplicada
i Telemàtica
Secció d'Enginyeria Telemàtica

UNIVERSITAT POLITECNICA DE CATALUNYA

PASCAL. PROGRAMAS

Departament de Matemàtica Aplicada
i Telemàtica
Secció d'Enginyeria Telemàtica

PROGRAMA GRAFO.-

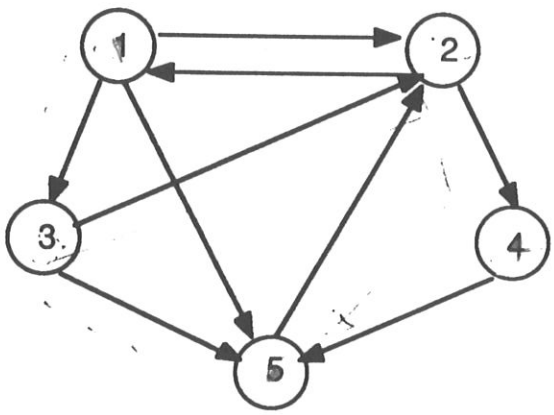
Enunciado.

En una gran ciudad tenemos N plazas unidas entre ellas por calles con un sentido de circulación establecido.

Crear un programa en PASCAL que nos muestre cuantos caminos posibles hay entre dos plazas (origen y destino), indicadas por el usuario, sin pasar dos veces por el mismo sitio.

A cada plaza le llamaremos nodo y le asignaremos un numero entero.

Ejemplo:



Realización

El programa generará una estructura, formada por variables dinámicas, que simule el grafo anterior.Esta estructura tendrá la siguiente composición (figura 1):

- Los primeros componentes de cada fila son las plazas que tiene la ciudad.
- A continuación de cada plaza se crea una lista enlazada con todos los nodos que están conectados directamente con dicha plaza.
- Los componentes de la primera columna estarán conectados entre si para facilitar el recorrido de la estructura.
- El número de nodos y las conexiones existentes entre ellos serán "infinitos"(el programa no debe presentar ningún tipo de restricción en cuanto al número se refiere. Este número sólo vedrá limitado por la capacidad de memoria de la máquina).

Para la busqueda de todos los caminos posibles entre una plaza (origen) y otra(destino) se realizará un procedimiento recursivo

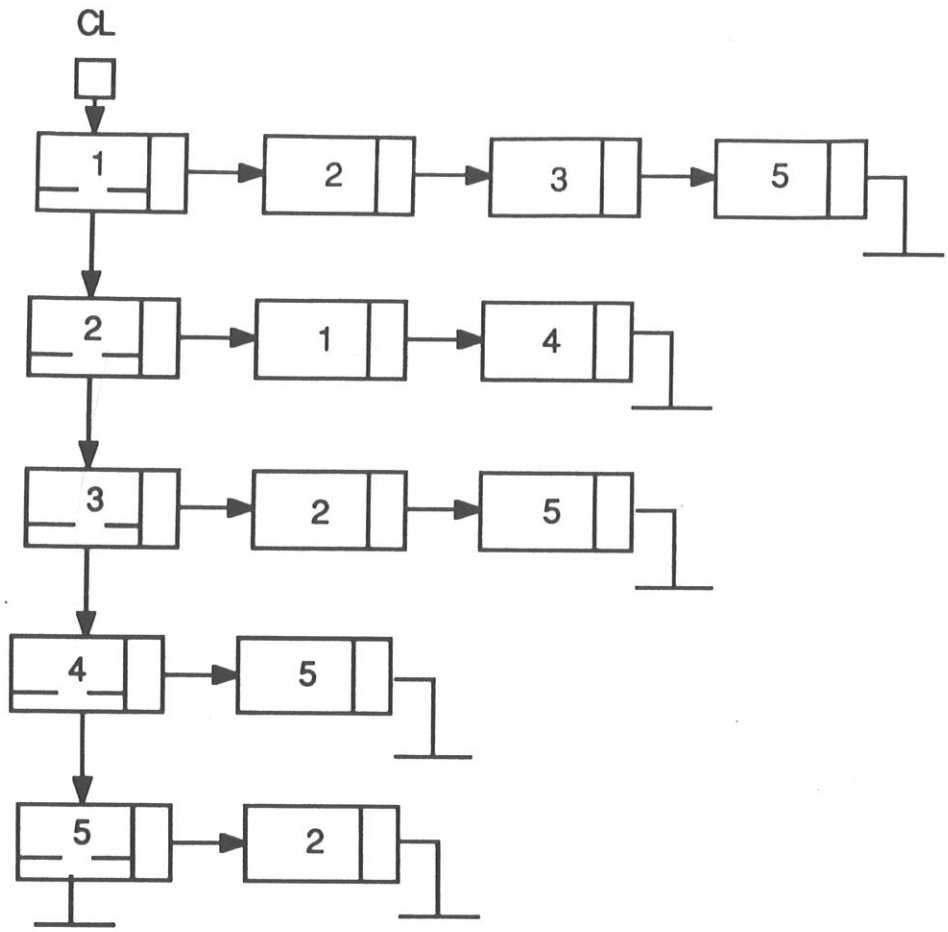


FIGURA 1

El resultado obtenido se guardará en una estructura como la anterior con la diferencia de que ahora cada fila indicará un camino distinto para ir desde el nodo origen(primer componente de cada fila) hasta el nodo destino(último componente de cada fila). Ejemplo:

Nodo origen : 1.

Nodo destino : 5.

La estructura que se debe generar es como la de la figura 2.

Para la creación de esta estructura se realizará un procedimiento (procedure creacion) que se propone como ejercicio. En dicho procedimiento la variable REPETIDO se pasa como parametro actual por variable desde el procedimiento BUSQUEDA.Su misión será detectar si el nodo que se pasa también como parámetro está repetido en el camino o si se trata del nodo destino.

Por último se deberá visualizar por pantalla el contenido de esta última estructura, que nos indicará todos los caminos posibles para ir desde un punto a otro.

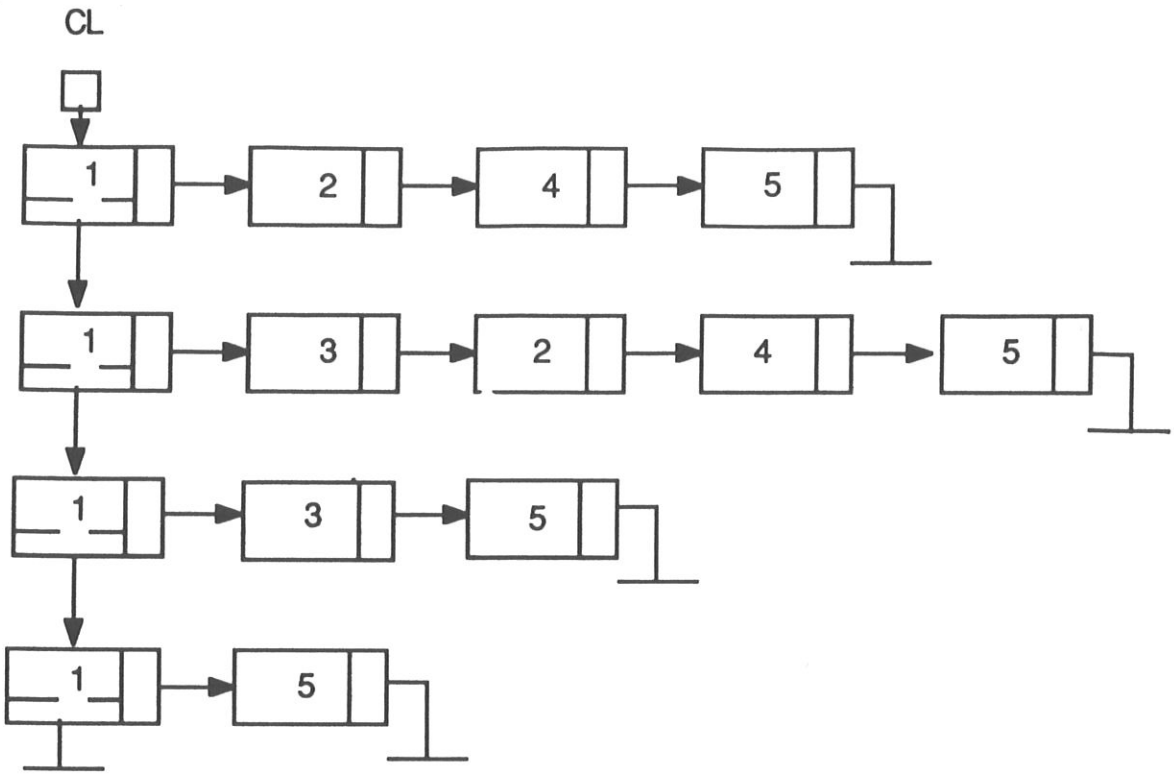


FIGURA 2

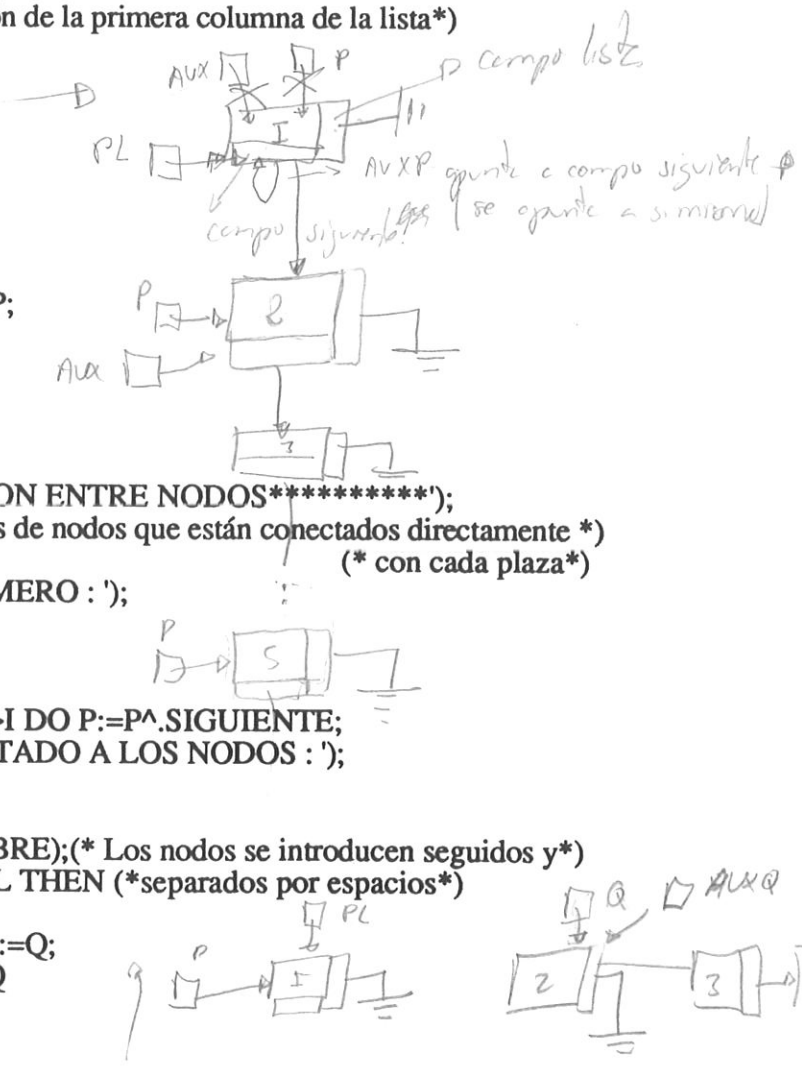
Programa.

```
PROGRAM GRAFO1(INPUT,OUTPUT);
TYPE
        PCALLE=^REGCALLE;
        PNUM=^REGNUM;
        REGNUM=RECORD
                NUMERO:INTEGER;
                SIGUIENTE:PNUM;
                LISTA:PCALLE
        END;
        REGCALLE=RECORD
                NOMBRE:PACKED ARRAY[1..20]OF CHAR;
                SIG:PCALLE
        END;
VAR
        CL:PNUM;

PROCEDURE SALIDA(VAR BL:PNUM);
(* Salida de datos por pantalla .Realiza la lectura de cualquier estructura creada en el
programa*)
VAR
        P:PNUM;
        Q:PCALLE;
BEGIN
        P:=BL;
```

```
        WHILE P<>NIL DO
        BEGIN
                WRITE(P^.NUMERO);
                Q:=P^.LISTA;
                WHILE Q<>NIL DO
                BEGIN
                        WRITE(Q^.NOMBRE);
                        Q:=Q^.SIG
                END;
                P:=P^.SIGUIENTE;
                WRITELN
        END
END;

PROCEDURE CREALISTA(VAR PL:PNUM);
(* Procedimiento que crea la estructura de variables dinámicas que representará el grafo
de la ciudad *)
VAR
        I,N:INTEGER;
        CONTESTAR:CHAR;
        AUXP,P:PNUM;
        AUXQ,Q:PCALLE;
BEGIN
        WRITE('INTRODUCE EL NUMERO DE NODOS :');
        READLN(N);
        FOR I:=1 TO N DO    (*Creación de la primera columna de la lista*)
        BEGIN
                NEW(P);
                P^.NUMERO:=I;
                IF I=1 THEN
                BEGIN
                        PL:=P;
                        AUXP:=P
                END;
                AUXP^.SIGUIENTE:=P;
                AUXP:=P;
                P^.LISTA:=NIL
        END;
        P^.SIGUIENTE:=NIL;
        WRITELN;
        WRITELN('********CONEXION ENTRE NODOS**********');
        REPEAT (* Creación de las listas de nodos que están conectados directamente *)
                P:=PL;                                          (* con cada plaza*)
                WRITE('EL NODO NUMERO : ');
                REPEAT
                        READLN(I)
                UNTIL I<=N;
                WHILE P^.NUMERO<>I DO P:=P^.SIGUIENTE;
                WRITE(' ESTA CONECTADO A LOS NODOS : ');
                REPEAT
                        NEW(Q);
                        READ(Q^.NOMBRE);(* Los nodos se introducen seguidos y*)
                        IF P^.LISTA=NIL THEN (*separados por espacios*)
                        BEGIN
                                P^.LISTA:=Q;
                                AUXQ:=Q
                        END
```



```
                    ELSE AUXQ^.SIG:=Q;
                    AUXQ:=Q;
                    Q^.SIG:=NIL
                UNTIL EOLN;
                WRITELN;
                WRITE('HAY MAS NODOS CON CONEXION ? (S/N) ');
                READLN;
                READLN(CONTESTAR)
        UNTIL CONTESTAR IN ['N','n']
END;

PROCEDURE  CREACION(.....);
(* Procedimiento que crea una estructura de variables dinámicas que representa todos los
caminos hallados en el procedimiento busqueda*)
BEGIN

        (* Ejercicio propuesto*)

END;'

PROCEDURE BUSQUEDA(NODOS:INTEGER);
(* Procedimiento encargado de realizar la busqueda de todos los caminos existentes entre
dos nodos*)
VAR
        BUSCA:PNUM;
        BUSCACALLE:PCALLE;
        REPETIDO:BOOLEAN;
BEGIN
        BUSCA:=CL;
        WHILE BUSCA^.NUMERO <> NODOS DO
        BUSCA:=BUSCA^.SIGUIENTE;
        BUSCACALLE:=BUSCA^.LISTA;
        WHILE BUSCACALLE <> NIL DO
        BEGIN
                CREACION(BUSCACALLE^,REPETIDO);
                IF NOT REPETIDO THEN BUSQUEDA(BUSCACALLE^.NOMBRE);
                BUSCACALLE:=BUSCACALLE^.SIG
        END;
END;
(* Programa principal*)

BEGIN
        CREALISTA(CL);
        WRITE('INTRODUCE EL NODO ORIGEN: ');
        READLN(ORIGEN);
        WRITE('INTRODUCE EL NODO DESTINO:');
        READLN (DESTINO);
        BUSQUEDA(ORIGEN);
        SALIDA

END.
```

PROGRAMA ECUACION 2º GRADO

Descripción del Algoritmo

- El Programa deberá presentar en pantalla las soluciones de una ecuación de 2 grado (a*x^2 + b*x + c = 0); dando al programa los coeficientes a,b,c.

- Según el valor de los coeficientes se obtiene distintos tipos de resultados

a = 0 - Solución de una ecuación de 1er. grado
Tres tipos de soluciones
b = 0 - No existe solución.
b,c = 0 - Solución para todo x.
b,c # 0 - Una unica solución x1 = -c/b

a # 0 - Hay tres tipos de soluciones según el discriminante (b^2 - 4*a*c) :

Discriminante > 0
Dos raices :
x1 =-b/a + sqrt(discriminante)/(2*a)
x2 =-b/a - sqrt(discriminante)/(2*a)

Discriminante = 0
Raiz doble x1,x2 = -b/a

Discriminante <0
Dos raices complejas conjugadas :
x1 =-b/a + i sqrt(discriminante)/(2*a)
x2 =-b/a - i sqrt(discriminante)/(2*a)

- Una vez realizado el cálculo de la solución, ésta deberá presentarse adecuadamente en pantalla las 6 tipos de soluciones.

```
(* Resolución de ecuaciónes de 1er. y 2 grado *)

program  Ecuaciones(input,ouput);

var
  a,b,c       : real;       (* Coeficientes de la ecuacion *)
  solu1,solu2 : real;       (* soluciones de la ecuacion *)
  tipoSolucion: integer;

(*--------------Entrada    de    datos-------------------------*)
procedure entrada ( var a,b,c : real);

 begin
    write('Entrar coeficientes a,b,c : ');
    readln(a,b,c)
 end;

(*--------Resolución de la  ecuación de primer grado-------*)
procedure Ecuacion1 (a,b : real; var cod:integer ; var solu :real );

 begin
   if b=0 then
      if b = 0 then
         cod := 0
       else
         cod := 1
    else
      begin
        cod := 2;
         solu := -c/b
      end
 end;

(*-------Resolución de la ecuación de segundo grado--------*)
procedure Ecuacion2 (a,b,c:real; var cod:integer; var x1,x2:real);

 var
   p,q,r : real;

   (*---- Cálculo del discriminante de la ecuación ----------*)
  function Disc(z,x,y:real):real;

  begin
       Disc:=sqr(z)-4*x*y
  end;

      (*--------Comienzo    del    procedimiento-----------------*)
begin
    p:= -b/(2*a);
  if disc(b,a,c) > 0  then
      begin
          q:=sqrt(d)/2*a;
        x1:=p+q;
        x2:=p-q;
       cod:=3
      end
   else
```

```
        if disc(b,a,c) = 0 then
           begin
             x1:=p;
            cod:=4
           end
        else
           begin
                  q:=sqrt(-disc(b,a,c))/(2*a);
             x1:=p;
             x2:=q;
            cod:=5
           end
 end;

(*------Presentación    de    resultados----------------------*)

procedure  Salida(tipoResultado:integer;  x1,x2:real);

 begin
  case TipoResultado of
     0 : writeln('Solución para todo x');
     1 : writeln('No existe solución');
   2 : begin
         writeln('Existe una raiz real simple : ');
         writeln(' x = ',x1:14:7)
       end;
   3 : begin
         writeln('Exite dos raices reales simples : ');
          writeln(' x1 = ',x1:14:7);
          writeln(' x2 = ',x2:14:7)
       end;
   4 : begin
         writeln('Existe una raiz real doble : ');
          writeln(' x1,x2 = ',x1:14:7)
       end;
   5 : begin
         writeln('Existen dos raices complejas conjugadas : ');
           writeln(' x1 = ',x1:14:7,' (p.real) ',x2:14:7,' (p.imag.) ');
           writeln(' x1 = ',x1:14:7,' (p.real) ',-x2:14:7,' (p.imag.) ');
      end
 end
end;


(*--------------Programa      Principal----------------------*)


begin
  entrada(a,b,c);
 if a=0 then
      Ecuacion1(b,c,tipoSolucion,Solu1)
   else
       Ecuacion2(a,b,c,tipoSolucion,Solu1,Solu2);
  salida(TipoSolucion,Solu1,Solu2)
end.
```

PROGRAMA LISTA ENLAZADA DOBLEMENTE.-

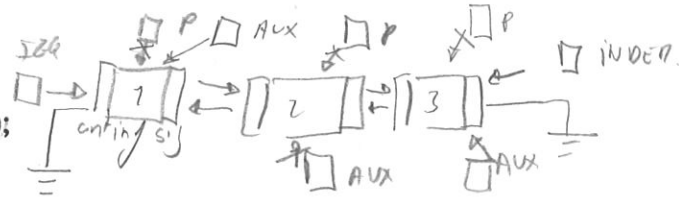
Enunciado

Crear un programa en lenguaje PASCAL que genere una lista de elementos enlazados entre ellos de tal forma que el acceso a dichos elementos se pueda realizar en cualquier sentido (de izquierda a derecha o de derecha a izquierda). Además el programa debe permitir la visualización de todo los elementos de la lista,en cualquira de los sentidos, o la visualización de un número indicado de ellos.

Programa

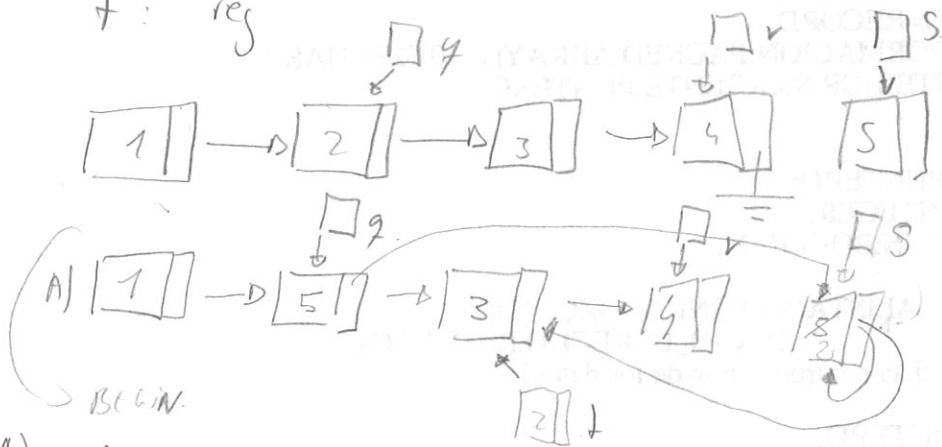
```
PROGRAM LISTADOBLE(INPUT,OUTPUT);
{Creacion de una lista de elementos ,del tipo registro, enlazada en los
 dos sentidos}
TYPE
        PUNTERO=^REGISTRO;
        REGISTRO=RECORD
                INFORMACION:PACKED ARRAY[1..10]OF CHAR;
                ANTERIOR,SIGUIENTE:PUNTERO
        END;
VAR
        IZQ,DER:PUNTERO;
        OPCION:INTEGER;
        PROTECCION:BOOLEAN;

PROCEDURE CREALISTA(VAR INICIO_IZQUIERDO,
                                INICIO_DERECHO:PUNTERO);
{ Generacion de la lista e introduccion de los datos}
VAR
        P,AUX:PUNTERO;
        RESPUESTA:CHAR;
BEGIN
        NEW(P);
        INICIO_IZQUIERDO:=P;
        P^.ANTERIOR:=NIL;
        AUX:=P;
        WRITE('INTRODUCE INFORMACION : ');
        READLN(P^.INFORMACION);
        WRITE('MAS ELEMENTOS ? (S/N) ');
        READLN(RESPUESTA);
        IF RESPUESTA IN ['S','s'] THEN
        REPEAT
                NEW(P);
                AUX^.SIGUIENTE:=P;
                P^.ANTERIOR:=AUX;
                AUX:=P;
                WRITE('INTRODUCE INFORMACION : ');
                READLN(P^.INFORMACION);
                WRITE('MAS ELEMENTOS ? (S/N) ');
                READLN(RESPUESTA)
        UNTIL RESPUESTA IN ['N','n'];
        P^.SIGUIENTE:=NIL;
        INICIO_DERECHO:=P
END;
```



N°6 Ejercicio de problemas.

```
TYPE
  APUN = ^REG;
  REG = RECORD
      INFO : INTEGER;
      SIG : Apun
  END;
VAR
  q, r : cpunt;
   s : ^reg;
   t : reg
```

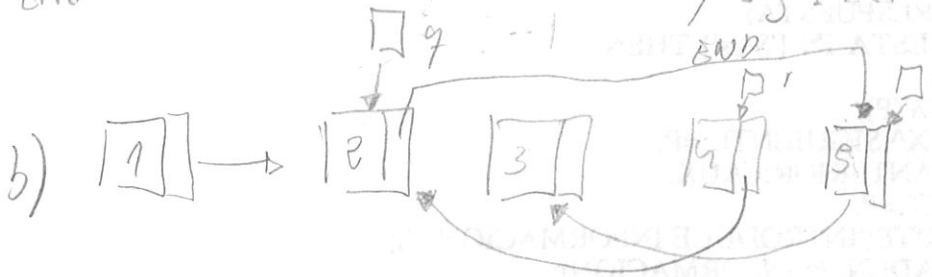


a) `s^.sig := s;` → el campo siguiente de donde apunta s le asignas

```
t := q^;
q^.sig := s^.sig;
s^ := t
END
```

```
BEGIN
r^.sig := q;
q := q^.sig^.sig^.sig;
s^.sig := q^.sig;
q^.sig := s
END
```

q no hace nada porque apunta a nil



```
PROCEDURE LEERLISTA (POINTER:PUNTERO;SENTIDO:CHAR);

{ Este procedimiento permite la visualización ,por pantalla, de los datos
contenidos en la lista doblemente enlazada. Es posible elegir el sentido
de lectura y el n·mero de componentes a visualizar}

VAR
        TODOS:CHAR;
        CONTADOR,CUANTOS:INTEGER;

BEGIN
       WRITE('DESEAS VER TODOS LOS COMPONENTES DE LA LISTA (S/N)?');
        READLN(TODOS);
        IF TODOS IN ['S','s'] THEN CUANTOS:=-1
        ELSE
                BEGIN
                        WRITE('CUANTOS ELEMENTOS QUIERES VER ? ');
                        READLN(CUANTOS)
                END;
        CONTADOR:=0;
        WRITELN('LOS COMPONENTES DE LA LISTA SON : ');
        WHILE ((POINTER<>NIL) AND ((CUANTOS<0)OR
                                        (CUANTOS<>CONTADOR))) DO
        BEGIN
                WRITE(POINTER^.INFORMACION);
                IF SENTIDO ='I' THEN POINTER:=POINTER^.SIGUIENTE
                ELSE POINTER:=POINTER^.ANTERIOR;
                CONTADOR:=CONTADOR+1
        END;
        WRITELN
END;


PROCEDURE MENU;

{ Procedimiento de presentación, por pantalla, de todas las opciones
que presenta el programa}

BEGIN

        WRITELN('OPCIONES : ');
        WRITELN('1.- LEER LA LISTA DE IZQUIERDA A DERECHA');
        WRITELN('2.- LEER LA LISTA DE DERECHA A IZQUIERDA');
        WRITELN('3.- CREAR NUEVA LISTA');
        WRITELN('4.- SALIR');
        WRITE('(Atención!! es necesario tener la lista creada para acceder ' );
        WRITELN('a las opciones 1,2)');
        WRITE('NUMERO DE LA OPCION : ')

END;
```

```
{programa principal}
BEGIN
        PROTECCION:=FALSE;
        REPEAT
                MENU;
                REPEAT
                        READLN(OPCION);
                UNTIL (OPCION IN [3,4]) OR PROTECCION;
                CASE OPCION OF
                        1:LEERLISTA(IZQ,'I');
                        2:LEERLISTA(DER,'D');
                        3:BEGIN
                                CREALISTA(IZQ,DER);
                                PROTECCION:=TRUE
                          END;
                END
        UNTIL OPCION=4
END.
```

A: SET OF [3,4]

TRATAMIENTO DE UN FICHERO DE PERSONAL

Se desea realizar un programa Pascal que contemple las siguientes opciones en un menú principal:

1 : Creación del Fichero de Personal
2 : Visualizar Fichero de Personal
3 : Añadir Registros
4 : Borrar Registro
5 : Finalizar

Los registros del fichero creado poseen 3 campos:
- Nombre (de 24 caracteres)
- Teléfono (número entero)
- Ciudad (de 24 carateres)

Para finalizar la entrada de datos en la opción 1, en la última petición de Nombre, se entrará únicamente el carácter [.]

En la visualización del fichero se mostrará la información de 20 en 20 registros, indicando al pié de pantalla que habrá de pulsarse Return para continuar la visualización.

Los registros entrados en la opción 3 se añadiran siempre al final del fichero existente, y la entrada de datos finalizará de igual forma que en la opción 1.

Para borrar un registro en la opción 4 se solicitará el campo Nombre del registro a borrar. Si el valor no se encuentra, el fichero quedará como estaba, y si se encuentra, se suprimen todos los registros con el campo nombre igual al valor entrado.

```
{-------------------------------------------------------------}
{          TRATAMIENTO DE UN FICHERO PERSONAL                 }
{-------------------------------------------------------------}
{  Se realiza la creacion y visualizacion del fichero, asi como el }
{  borrado y añadido de registros. Los registros son de tres campos: }
{  Nombre y apellidos / Telefono / Ciudad                      }
{-------------------------------------------------------------}

Program  Fichero_de_Personal(Input,Output,F,G);

Type Cadena24 = Packed Array[1..24] of Char;
     Reg_Personal = Record
                      Nombre:Cadena24;
                       Telefono:LongInt;
                      Ciudad:Cadena24
                    End;
     F_Personal = File Of Reg_Personal;
Var   F,G : F_Personal;
      I : Integer;
     C : Char;

{-------------------------------------------------------------}
{ Lectura de Packed Array de caracteres                        }
{-------------------------------------------------------------}

Procedure Leer(Var Pal:Cadena24; Lon:Integer);
     Var I:Integer;
    Begin
      For I:=1 To Lon Do
       If Eoln Then Pal[I]:=' '
             Else Read(Pal[I]);
      Readln
   End;
```

```
{-------------------------------------------------------------}
{ Creacion del Fichero de Personal PERSONAL.DAT               }
{ Si ya existia, los datos anteriores se borraran             }
{-------------------------------------------------------------}

Procedure Creacion(Var F:F_Personal);
  Var R : Reg_Personal;

  Begin
     Rewrite(F);
     Writeln;
   Writeln('CREACION DEL FICHERO DE PERSONAL:');
     Writeln;
     Writeln('(Finalizar con: Nombre  = .  )');
     Writeln;
    Write('Nombre  = ');
    Leer(R.Nombre,24);
   While R.Nombre[1]<>'.' Do
    Begin
       Write('Telefono = ');
      Readln(R.Telefono);
      Write('Ciudad = ');
      Leer(R.Ciudad,24);
      Write(F,R);
      Write('Nombre  = ');
      Leer(R.Nombre,24)
   End
End;
```

```
{-------------------------------------------------------------}
{ Visualizacion del contenido del Fichero PERSONAL.DAT        }
{-------------------------------------------------------------}

Procedure Visualizar(Var F:F_Personal);
 Const Cabecera1 ='Nombre y Apellidos        Telefono   Ciudad       ';
               Cabecera2 ='------------------------   --------   -------------';
 Var R : Reg_Personal;
     Lin:Integer;
 Begin
  Reset(F);
  Lin:=0;
   Writeln;
  Writeln('LISTADO DE PERSONAL ACTUAL:');
   Writeln;
   Writeln(Cabecera1);
   Writeln(Cabecera2);
  While Not Eof(F) Do
   Begin
    Read(F,R);
     Lin:=Lin+1;
     If Lin=20
      Then Begin
            Lin:=0;
             Writeln;
               Write('-- Continue con <Return> --');
            Readln;
             Writeln;
              Writeln(Cabecera1);
              Writeln(Cabecera2);
          End;
       Writeln(R.Nombre,R.Telefono:9,R.Ciudad:28)
   End;
   Writeln;
   Write('-- Continue con <Return> --');
  Readln
End;
```

```
{-------------------------------------------------------------}
{ Copiar PERSONAL.DAT en un fichero auxiliar. Esto permitira añadir }
{ mas registros al final de los ya existentes en PERSONAL.DAT }
{-------------------------------------------------------------}

  Procedure Copiar(Var G,F:F_Personal);
  Var R:Reg_Personal;
  Begin
    Reset(F);
     Rewrite(G);
    While Not Eof(F) Do
     Begin
      Read(F,R);
       Write(G,R);
     End;
  End;

{-------------------------------------------------------------}
{ Entrada de los datos de personal en registros que se añaden al }
{ final de los ya existentes en PERSONAL.DAT }
{-------------------------------------------------------------}
Procedure AnadirReg(Var G,F:F_Personal);
  Var R : Reg_Personal;

  Begin
    Copiar(G,F);
     Writeln;
   Writeln('ADICION DE REGISTROS AL FICHERO DE PERSONAL:');
     Writeln;
     Writeln('(Finalizar con: Nombre  = .  )');
     Writeln;
    Write('Nombre  = ');
    Leer(R.Nombre,24);
   While R.Nombre[1]<>'.' Do
    Begin
       Write('Telefono = ');
      Readln(R.Telefono);
      Write('Ciudad = ');
      Leer(R.Ciudad,24);
      Write(G,R);
      Write('Nombre  = ');
      Leer(R.Nombre,24)
   End;
   Copiar(F,G)
End;
```

```
{-------------------------------------------------------------}
{ Borrar un registro a partir de su designacion en PERSONAL.DAT }
{-------------------------------------------------------------}

Procedure BorrarReg(Var G,F:F_Personal);
Var R : Reg_Personal;
   Nom : Cadena24;
Begin
Writeln;
Writeln('BORRADO DE UN REGISTRO EN FICHERO DE PERSONAL:');
Writeln;
Write('Nombre del registro a borrar : ');
Leer(Nom,24);
Reset(F);
Rewrite(G);
While Not Eof(F) Do
  Begin
    Read(F,R);
    If R.Nombre<>Nom Then Write(G,R)
  End;
  Copiar(F,G)
End;
```

```
{------------------------------------------------------------}
{ Programa Principal: Seleccion de opciones para el mantenimiento }
{                     del fichero PERSONAL.DAT                 }
{------------------------------------------------------------}

Begin
  Assign(F,'PERSONAL.DAT');          { Sólo para Turbo-Pascal }
  Assign(G,'TEMPORAL.DAT');          { Sólo para Turbo-Pascal }
  Repeat
    Repeat
      Writeln;
     Writeln('MANTENIMIENTO DEL FICHERO DE PERSONAL');
                 Writeln('-----------------------------------');
       Writeln('1 : Creación del Fichero de Personal');
       Writeln('2 : Visualizar Fichero de Personal');
       Writeln('3 : Añadir Registros');
       Writeln('4 : Borrar Registro');
       Writeln('5 : Finalizar');
      Writeln;
      Write('    Indicar Opcion elegida : ');
     Readln(C)
    Until C In ['1'..'5'];
   Case C Of
    '1' : Creacion(F);
     '2' : Visualizar(F);
    '3' : AnadirReg(G,F);
    '4' : BorrarReg(G,F)
   End
  Until C='5';
  Close(F)
End.
```

PROGRAMA JUEGO DE LA VIDA:

Se desea diseñar un programa que modeliza la evolución de una colonia de células siguiendo las reglas del "juego de la vida".
Cada generación de la colonia de células habita en una matriz cuadrada, de L filas y L columnas (2<L<=20) de elementos boolean, donde los índices indican la localización de la célula en la matriz, siendo el valor de la célula True si está viva o False si no hay célula o está muerta. Un indice adicional (de valor 0 ó 1) se utilizará para identificar la generación actual y la siguiente, únicas que precisan ser almecenadas para calcular cada nueva generación, en la forma siguiente:

Una célula nace (en la siguiente generación), si en la generación actual su casilla está rodeada de exactamente 3 células vivas.

Toda célula rodeada de 2 ó 3 células vivas sobrevive en la generación siguiente.

Toda célula rodeada de 0 ó 1 celulas muere de inanición en la siguiente generación, y si está rodeada de 4 o más células, muere asfixiada.

Cualquier casilla tiene 8 casillas adyecentes que la rodean. Esto de aplica también a las casillas situadas en los bordes de la matriz: Las casillas de la primera y última fila son adyacentes entre sí, al igual que las casillas de la primera y última columna. También son adyacentes entre sí las cuatro casillas de las cuatro esquinas de la matriz.

El programa solicitará los valores de la generación inicial. Al mostrar cada una de las generaciones siguientes, se solicitará si se desea finalizar la salida o se desea continuar.

En la entrada y la visualización, se emplará el carácter [.] para celda vacía y otro carácter distinto, por ejemplo [*] para célula viva.

```
{-------------------------------------------------------}
{ PROGRAMA JUEGO DE LA VIDA                             }
{  Se modeliza la vida de una colonia de "celulas vivas" }
{  mediante una matriz cuadrada de elementos boolean.    }
{  Cada elemento de la matriz corresponde a una celula. Si }
{  su valor es TRUE ello indica que la celula esta viva. Si }
{  es FALSE indica que no existe celula o que esta muerta }
{  Mediante unas reglas de nacimiento, supervivencia y muerte }
{  de las celulas, que dependen del estado de las celulas }
{  "contiguas", se obtienen las sucesivas generaciones.  }
{-------------------------------------------------------}
program  Vida(Input,Output);
uses crt;  {--> Solo para Turbo-Pascal Version 4 o 5 }
const  max=21;
type Gener=array[0..1,1..Max,1..Max] of boolean;
var cel:Gener;
    c:char;
     k,ks,L,ngen,ng,ngs:integer;

{-------------------------------------------------------}
{ Entrada de valores para la generacion inicial de la colonia }
{ (generacion 0). El tamaño del cuadro L indica el numero de }
{ filas y columnas de la matriz. Se entrara un [.] para una }
{ celula vacia, y otro signo, por ejemplo [*], para una celula }
{ viva.                                                 }
{-------------------------------------------------------}
procedure InicialGen(var cel:Gener; L:integer);
var  i,j:integer;
    c:char;
begin
   writeln; writeln('GENERACION: 0'); writeln;
  for i:=0 to L+1 do
      for j:=0 to L+1 do
           cel[0,i,j]:=False;      { Inicialización: colonia vacía }
  for i:=1 to L do
  begin
     for j:=1 to L do
     begin
         if eoln then c:='.'
                 else read(c);
         cel[0,i,j]:=c<>'.'
     end;
     readln
  end
end;
```

```
{-------------------------------------------------------}
{  Calculo de la nueva generacion a partir de la generacion  }
{  anterior. Ambas se diferencian en el primer indice del  }
{  array Cel: k=Generacion anterior, ks=Nueva generacion.  }
{  Es suficiente que este indice tome solo valores 0 o 1,  }
{  alternando convenientemente dichos valores. De esta forma  }
{  la memoria empleada sera muy inferior a la empleada si se  }
{  le asignaran tantos valores como generaciones se desean  }
{  obtener.                                                 }
{-------------------------------------------------------}
procedure NuevaGen(var cel:Gener; k,ks:integer);
   var  i,j,ie,je,nv,p,q:integer;

   function  ajuste(n:integer):integer;
  begin
     ajuste:=n;
     if n>L then ajuste:=1
            else if n<1 then ajuste:=L
  end;


begin
  for i:=1 to L do
    for j:=1 to L do
    begin        { i y j determinan la posicion de una celda a estudiar }
     nv:=0;
      for p:=-1 to 1 do          { p,q = valores a sumar a i,j para acceder a }
        for q:=-1 to 1 do         { las celulas contiguas a la i,j. Excepcion  }
         if not((p=0) and (q=0))   { para p=q=0                          }
            then begin
                    ie:=ajuste(i+p);
                    je:=ajuste(j+q);
                    nv:=nv+ord(cel[k,ie,je]);
                     if nv=3 then cel[ks,i,j]:=true
                             else if nv=2 then cel[ks,i,j]:=cel[k,i,j]
                                          else cel[ks,i,j]:=false
              end
   end
end;
```

```
{-------------------------------------------------------}
{ Se escriben en pantalla los valores de los elementos de la  }
{ matriz de la generacion ngs.                                }
{-------------------------------------------------------}
procedure EscribirGen(cel:Gener; ks,ngs:integer);
   var i,j:integer;
     c:char;
begin
  writeln('GENERACION: ',ngs); writeln;
  for i:=1 to L do
  begin
    for j:=1 to L do
    begin
       if cel[ks,i,j] then c:='*'
                  else c:=' ';
      write(c:2)
   end;
    writeln
  end;
   writeln
end;


{-----------------------------------------}
{ Programa principal                      }
{---------------------      ---------------------}
Begin
  repeat
    write('Tamaño del cuadro:.....'); readln(L)
  until L>2;
   InicialGen(cel,L);
  ng:=0;
  repeat
   k:=ng mod 2;
   ngs:=ng+1;
   ks:=ngs mod 2;
    NuevaGen(cel,k,ks);
    write('>> Return=Continuar F=Final  :'); C:=readkey;
   clrscr; {--> SOLO PARA TURBO-PASCAL VERSION 4 O 5}
     EscribirGen(cel,ks,ngs);
   ng:=ng+1
  until c in ['f','F']
end.
```

PROGRAMA PALABRAS

Se desea obtener un programa Pascal que contemple las siguientes opciones en un menú principal:

1. Creación de un fichero de texto. Las líneas de texto del mismo se introducirán mediante el teclado, finalizando cada línea con la tecla de Return, y finalizando el texto con la secuencia Return seguido de Ctrl-Z.

2. Visualización en pantalla del fichero existente, creado en el punto anterior.

3. Obtener y mostrar en pantalla el número total de palabras existentes en el texto. Se supone que una palabra es cualquier secuencia de caracteres, excluido el espacio en blanco. Dos palabras estarán separadas al menos por un espacio en blanco o un carácter de fin de línea.

4. Finalizar.

LISTADO:

```
{-------------------------------------------------------}
{                  Programa Palabras                    }
{                                                       }
{     Se realiza la creacion y visualizacion de un fichero }
{     de texto, que debe contener como primer caracter de  }
{     la ultima linea la marca de fin de fichero (Ctrl-Z)  }
{     Permite contar el número de "palabras" que contiene  }
{    el fichero creado.                                   }
{-------------------------------------------------------}
program  palabras(input,output,f);

var  f:text;
     n:integer;
     opcion:integer;

{-------------------------------------------------------}
{     Creacion del fichero de texto: F.DAT              }
{-------------------------------------------------------}
procedure  crear(var  f:text);
var c:char;
begin
    rewrite(f);
  while not eof do
  begin
    while not eoln do
    begin
      read(c);
        write(f,c)
   end;
  readln;
    writeln(f)
 end
end;
```

```
{-------------------------------------------------------------}
{     Visualizacion del fichero de texto: F.DAT              }
{-------------------------------------------------------------}
procedure visualizar(var f:text);
var c:char;
begin
   reset(f);
  while not eof(f) do
  begin
    while not eoln(f) do
   begin
     read(f,c);
      write(c)
   end;
    readln(f);
    writeln
 end
end;

{-------------------------------------------------------------}
{    Recuento de palabras del fichero F.DAT                   }
{     Se utilizan dos variables Char: C = caracter acabado de leer, y   }
{     CANT = caracter anterior leido. Después de cada fin de línea, se  }
{     asigna a C el espacio en blanco. Antes de leer un nuevo valor de C }
{     se asigna a CANT el valor de C. Se cuenta una nueva palabra cada  }
{     vez que C es distinto del caracter blanco y CANT es un blanco     }
{-------------------------------------------------------------}
procedure contar(var f:text; var p:integer);
var c,cant:char;
begin
c:=' ';
p:=0;
reset(f);
while not eof(f) do
  begin
  while not eoln(f) do
   begin
     cant:=c;
      read(f,c);
     if (c<>' ') and (cant=' ') then p:=p+1
   end;
  readln(f);
 c:=' '
 end
end;
```

```
{-------------------------------------------------------------}
{    Programa principal                                       }
{-------------------------------------------------------------}
Begin
assign(f,'texto.dat');
repeat
  repeat
     writeln('1: Crear fichero texto');
     writeln('2: Visualizar fichero existente');
     writeln('3: Contar palabras del fichero existente');
     writeln('4: Finalizar');
    writeln;
    write('Indique opcion elegida = ');
    readln(opcion)
  until opcion in [1..4];
  case opcion of
   1: begin
         writeln('Escriba las líneas de texto. Para finalizar: CTRL-Z');
                  writeln('-----------------------------------');
       crear(f);
                  writeln('-----------------------------------');
     end;
   2: begin
         writeln('El contenido del texto existente es:');
                  writeln('-----------------------------------');
       visualizar(f);
                  writeln('-----------------------------------')
     end;
   3: begin
       contar(f,n);
                 writeln('-------------------------');
       writeln('Número de palabras = ',n);
                 writeln('-------------------------')
     end
 end
until opcion=4;
close(f)
End.
```

TORRES DE HANOI

Enunciado

El problema de las torres de Hanoi consiste en tres varillas 1,2,3. Una de ellas atraviesa N discos de tal forma que cada disco es más pequeño que el que se encuentra inmediatamente debajo de él. Se pretende realizar un programa que nos diga de que forma podemos pasar los N discos colocados en la varilla 1, por ejemplo, a la varilla 3 cumpliendo las siguientes condiciones:

.- Sólo podemos pasar un disco en cada movimiento.

.- No podemos colocar un disco encima de otro más pequeño.

.- La varilla 2 se puede utilizar para realizar los pasos intermedios

Solución

El programa se soluciona utilizando recursividad siendo la idea de funcionamiento la siguiente:

- Mover N-1 discos de la varilla 1 a la 2.
- Mover un disco de la varilla 1 a la varilla 3
- Mover los N-1 discos de la varilla 2 a la 3.

Programa

```
PROGRAM HANOI (INPUT,OUTPUT);
VAR
        NDISCO:INTEGER;

PROCEDURE TORRE(NDISCO,ORIGEN,DESTINO,CENTRO:INTEGER);
BEGIN {de torre}
        IF NDISCO > 0 THEN
        BEGIN
                TORRE(NDISCO-1,ORIGEN,CENTRO,DESTINO);
                WRITELN('DESPLAZAR EL DISCO
                                DE',ORIGEN:3,'A',DESTINO:3);
                TORRE(N-1,CENTRO,DESTINO,ORIGEN)
        END
END; {de torre}
BEGIN {principal}
        READ(N);
        WRITELN('LA SOLUCION DEL PROBLEMA DE LAS TORRES DE
                        HANOI');
        WRITELN('PARA ',N:4,'DISCO ES:');
        TORRE(N,1,3,2)
END.  {principal}
```

PROGRAMA HORARIOS

Diseñar un programa Pascal que contemple las siguientes opciones en un menú principal:

1. Crear un fichero, conteniendo la siguiente información en cada registro:
   - Nombre de persona (30 caracteres).
   - Horario de trabajo: Campo en forma de array de conjuntos. El índice del array será el día de la semana (valor del tipo enumerado), y cada componente del mismo será un conjunto de horas, del tipo base subrango 8..20.
     El horario se entrará indicando en primer lugar el número total de horas de trabajo semanales, y a continuación, el día (entrado en forma numérica) y la hora correspondiente.

   Después de la entrada de cada registro, se indicará si quedan más personas por entrar.
2. Visualizar los horarios de las personas existentes en el fichero entrado:
   A partir de la variable array del horario de cada persona, donde cada elemento es un conjunto de horas, se escribe una tabla en pantalla, indicando un fila para cada dia, mostradose las horas correspondientes de cada día de isquierda a derecha. Si la hora está en el conjunto, se ecribe su valor, y si no, se escriben guiones en su lugar
3. A partir de los horarios de las personas existentes, se calcula el horario en que todas las personas están trabajando (intersección de las horas de trabajo), mostrando dicho horario resultante de igual forma que en la opción 2.
4. Calcular el horario en que todas las personas existentes están simultáneamente en "horas libres" (horas en que no trabajan, visualizando el resultado de la misma forma que en caso anterior.
5. Finalizar la ejecución del programa.

LISTADO:

```
{-------------------------------------------------------------------------}
{  PROGRAMA HORARIO:                                                      }
{  Se genera un fichero de registros, con dos campos: Nombre y            }
{  Horario, el cual consiste en un array (un componente para              }
{  cada dia, de Lunes a Viernes), siendo cada componente un               }
{  conjunto (Set) de horas (de tipo base 8..20).                          }
{  El programa muestra como puede operarse con esta estructura,           }
{  con varias opciones seleccionables por menu:                           }
{  1. Crear el fichero, conteniendo una lista de personas con             }
{     su horario respectivo                                               }
{  2. Visualizar horarios de las personas del fichero existente           }
{  3. Hallar el horario de coincidencia de horas ocupadas, para           }
{     todas las personas del fichero existente                            }
{  4. Hallar el horario de coincidencia de horas libres, para             }
{     todas las personas del fichero existente                            }
{  5. Final del programa.                                                 }
{  El programa podria ampliarse incluyendo mas informacion de             }
{  cada persona, como diferentes tipos de horarios, localizacion          }
{  (numero de despacho, etc.).                                            }
{  Tambien podrian ampliarse las opciones del menu principal,             }
{  como pueden ser: modificacion de horarios existentes, añadir           }
{  o borrar personas en el fichero, consultar el horario de una           }
{  persona determinada, etcetera.                                         }
{                                                                         }
{-------------------------------------------------------------------------}
Program Horario(Input,Output,F);

Const Lon    = 30;
Type  Cadena = Packed Array[1..Lon] of Char;
      RangoH = 8..20;
      Horas  = Set Of RangoH;
      Tdia   = (Lu,Ma,Mi,Ju,Vi);
      Dias   = Array[Tdia] of Horas;
      Datos  = Record
                 Nom  : Cadena;
                 Hor  : Dias;
               End;
      Ficher = File of Datos;

Var  F     : Ficher;
     Opcion : Char;

{-------------------------------------------------------------------------}
{  Lectura de un Packed Array de caracteres, valida para                  }
{  cualquier compilador de Pascal                                         }
{-------------------------------------------------------------------------}
Procedure Leer(Var N:Cadena);
 Var K     : 1..Lon;
 Begin
  For K:=1 to Lon do
   If Not Eoln Then Read(N[K])
               Else N[K]:=' ';  {LLena con "blancos" los }
  Readln                        { caracteres sobrantes  }
 End;
```

```
{-------------------------------------------------------------------------}
{  Creacion y entrada de datos en el fichero 'F.DAT' que contiene }
{  los Nombres y Horarios de cada persona. Finaliza la entrada    }
{  cunado se responda que no quedan más personas por entrar       }
{-------------------------------------------------------------------------}
Procedure Llenar_Lista(Var F:Ficher);
 Var I     : Integer;
    D     : Tdia;
    Hora,
    K,NH   : Integer;
    C     : Char;
    R     : Datos;

 {-------------------------------------------------------------------}
 {  Procedimiento local a Llenar_Lista:                             }
 {  Dado que el indice Dia utilizado es un tipo enumerado,          }
 {  debe utilizarse una entrada indirecta de estos valores, en      }
 {  este caso la entrada de valores por teclado es numerica         }
 {  obteniendose en este procedimiento el valor adecuado de la      }
 {  variable D de tipo enumerado                                    }
 {-------------------------------------------------------------------}

 Procedure Entrar_Dia(Var D:TDia);
  Var NDia : Integer;
  Begin
     Repeat
      Writeln('Dias: 1=Lun 2=Mar 3=Mie 4=Jue 5=Vie');
      Writeln;
       Write('Entrar dia 1..5 = ? ');
      Readln(Ndia)
     Until NDia In [1..5];
     Case NDia Of
      1 : D:=Lu;  2: D:=Ma;  3: D:=Mi;  4: D:= Ju;  5: D:=Vi
     End
  End;

 Begin
  Rewrite(F); {--> Prepara el fichero F para escritura. Si ya existia   }
  I:=0;          { previamente, la informacion que contenia se pierde   }
  Repeat         { A continuacion se realiza la entrada de datos  en    }
   I:=I+1;      { los campos de la variable R (registro del fichero)    }
    Writeln('Datos Persona ',I,' :'); Writeln;
    Write('Nombre = ? ');
    Leer(R.Nom);
    Writeln('Horario:');
    Writeln;
    Repeat
     Write('Numero de horas = ? ');
     Readln(NH)
    Until NH in [0..65];
    Writeln;

    {Continua pag. siguiente}
```

```
    For D:=Lu to Vi do R.Hor[D]:=[]; { El conjunto de horas de cada dia }
    For K:=1 to NH do                { se inicializa al conjunto vacio     }
    Begin
     Entrar_Dia(D);
     Repeat
       Write('Entrar hora 8..20 = ? ');
      Readln(Hora)
     Until Hora in [8..20];
     Writeln; Writeln;       { Cada uno de los valores de Hora entrado }
     R.Hor[D]:=R.Hor[D]+[Hora] { se añade al conjunto del dia D corresp. }
    End;
    Write(F,R); { Escritura del valor de R en el fichero F }
    Repeat
     Write('Quedan mas personas ? [s/n] ');
     Readln(c)
      Until c in ['s','n','S','N']
   Until c in ['n','N'];
  close(f)     { Final de la escritura en el fichero F }
 End;

{-------------------------------------------------------------------}
{  A partir del valor de la variable D (dia) tipo enumerado       }
{  este procedimiento escribe en pantalla las iniciales del       }
{  dia correspondiente                                            }
{-------------------------------------------------------------------}
Procedure Escribir_Dia(D:Tdia);
Begin
 Case D Of
   Lu : Write('Lu:');
   Ma : Write('Ma:');
   Mi : Write('Mi:');
   Ju : Write('Ju:');
   Vi : Write('Vi:')
 End
End;

{-------------------------------------------------------------------------}
{  A partir de la variable array Horario, donde cada elemento es        }
{  un conjunto (Set) de horas, se escribe una tabla en pantalla,        }
{  un fila para cada dia, indicando las horas correspondientes          }
{  del conjunto. Si la hora esta en el conjunto, se ecribe su           }
{  valor, y si no, se escriben guiones en su lugar                      }
{-------------------------------------------------------------------------}
Procedure Mostrar_Horario(Horario:Dias);
 Var D    : Tdia;
   Hora   : RangoH;
 Begin
   For D:=Lu to Vi do
   Begin
    Escribir_Dia(D);
    For Hora:=8 to 20 do
     If Hora In Horario[D] Then Write(Hora:3)
                           Else Write('--':3);
    Writeln
   End;
   Writeln
 End;
```

```
{-------------------------------------------------------------------------}
{ Haciendo uso del procedimiento anterior, se muestran los nombres }
{ y horarios de todas las personas del fichero F existente. Para }
{ evitar el Scroll de pantalla (desplazamiento hacia arriba del }
{ texto mostrado) se detiene la salida cada dos registros }
{-------------------------------------------------------------------------}
Procedure Mostrar_Lista(Var F:Ficher);
 Var R : Datos;
    K : Integer;
 Begin
  Reset(F);          {--> Prepara el fichero F para lectura }
  k:=0;
  While not eof(F) do
  Begin
   Read(F,R);        {--> Lectura del registro R del fichero F }
   Writeln;
   Writeln('Nombre:   ',R.Nom);
   Writeln;
   Writeln('Horario:');
   Mostrar_Horario(R.Hor); { Visualizacion del horario contenido en R }
   k:=k+1;                 { Contador de registros leidos = k          }
   if k mod 2 = 0 then begin { Detención de salida si k es par }
                      write('< Pulse Return para continuar >');
                      Readln
                     end;
  End
 End;

{-------------------------------------------------------------------------}
{ A partir de los horarios existentes, en forma de conjuntos, se }
{ calcula el producto de todos ellos, resultando asi el horario }
{ formado por las horas en que todos coinciden }
{ El conjunto sobre el que se multiplica debe ser inicializado al }
{ conjunto total (todos los elementos del tipo base: 8..20) }
{-------------------------------------------------------------------------}
Procedure Horas_Ocup(Var F:Ficher);
 Var Coinc : Dias;  { Conjunto donde se obtendra la interseccion }
    D    : Tdia;
    R    : Datos;
 Begin
    Reset(F);
    Writeln('Horas ocupadas coincidentes:');
    Writeln;
    For D:=Lu to Vi do Coinc[D]:=[8..20];  { Inicializacion a conjunto }
     While not eof(F) do                   { vacio para todos los dias }
     Begin
       Read(F,R);
       For D:=Lu to Vi do
         Coinc[D]:=Coinc[D]*R.Hor[D]  { Calculo de la interseccion }
     End;                             { (producto) de los horarios }
    Mostrar_Horario(Coinc)          { Muestra el horario resultante }
 End;
```

```
{-------------------------------------------------------------------------}
{ Procedimiento similar al anterior, pero para obtener la interseccion    }
{ de "horas libres" entre todas las personas del fichero. En este caso    }
{ las horas libres se obtienen por "diferencia" entre el cojnuto total     }
{ y las horas "ocupadas" (las que figuran en el conjunto). En este caso}
{ el conjunto sobre el que se hace el producto debe ser inicializado al   }
{ conjunto total 8..20                                                    }
{-------------------------------------------------------------------------}
Procedure Horas_Lib(Var F:Ficher);
 Var Coinc  : Dias;
     D      : Tdia;
     R      : Datos;
 Begin
    Reset(F);
    Writeln('Horas libres coincidentes:');
    Writeln;
    For D:=Lu to Vi do Coinc[D]:=[8..20];       { Inicializacion }
     While not eof(F) do
     Begin
       Read(F,R);
       For D:=Lu to Vi do
         Coinc[D]:=Coinc[D]*([8..20]-R.Hor[D])   { Interseccion }
     End;
    Mostrar_Horario(Coinc)       { Muestra el horario resultante }
 End;

{-------------------------------------------------------}
{ Procedimiento para mostrar el menu principal  }
{ y obtener la opcion seleccionada              }
{-------------------------------------------------------}
Procedure Menu(Var op:char);
 Begin
   Writeln;
   Writeln('MENU:');
   Writeln;
   Writeln('1: Entrar Horarios');
   Writeln('2: Mostrar Horarios');
   Writeln('3: Mostar Horas ocupadas coincidentes');
   Writeln('4: Mostrar Horas libres coincidentes');
   Writeln('5: Finalizar');
   Writeln;
   Repeat
     Write('Indique opccion elegida = ');
     Readln(op)
    Until op in ['1'..'5'];
   Writeln
 End;
```

```
{-------------------------------------------------}
{  Programa Principal. Permite la seleccion       }
{  repetitiva de opciones del menu principal      }
{-------------------------------------------------}
Begin
Assign(F,'F.DAT');
Repeat
 Menu(Opcion);
 Case Opcion Of
 '1': Llenar_Lista(F);
 '2': Mostrar_Lista(F);
 '3': Horas_Ocup(F);
 '4': Horas_Lib(f)
 End
Until Opcion='5'
End.
```

PROGRAMA TRIANGULO

ENUNCIADO: Dados los tres lados de un triángulo calcular su área. Es un ejemplo sencillo de una función booleana, que verifica si tres valores dados pueden ser lados de un triángulo La función utiliza un procedimiento local para intercambiar dos valores, utilizado en el proceso de ordenación de tres números, como caso particular del método de "burbuja" para la ordenación de secuencias.

```
program triangulo (input,output);

var a,b,c,s,sup : real;

{----------------Función Triangulo----------------}
function trian (x,y,z : real) : boolean;

  {------------Procedure Inter--------------------}
  procedure inter (var p,q : real);

  var  r : real;
  begin
    r := p ; p := q ; q := r
  end;

begin
  if x < y then inter (x,y);
  if y < z then inter (y,z);
  if x < y then inter (x,y);
  trian :=  ( x  < y+z ) and (z > 0)
end;

{------------{(* Programa Principal *)------------}

begin

  repeat
     write (' Entrar los tres lados del triangulo : ');
     readln(a,b,c)
  until trian(a,b,c);

  s:= (a+b+c)/2;
  sup := sqrt(s*(s-a)*(s-b)*(s-c));
  writeln (' El area del triangulo  es = ',sup:8:3);
end.
```

METODE D'ORDENACIO DIRECTE.

Aquest problema consta d'ordenar de menor a major una llista de dades estructurades, puntualment un array de N elements.

El mètode que farem servir s'anomena Mètode D'ordenació Directe i el seu algoritme és el següent:

Sigui X un array de N elements del tipus Integer :
Es tracte de comparar el valor X[i] amb la resta de valors de l'array -> X[j] per j = [i+1..N], i per i = [1..N].

Una vegada es compleix que X[j] < X[i] , ens trobem en què hi ha un element de l'array d'índex j > i que és menor a l'element X[i], llavors s'ha de fer un canvi d'ordre dins l'array.

Per realitzar aquest canvi d'una manera correcte es farà servir una variable integer auxiliar 'x' i un índex auxiliar 'K', en el què contindran el valor i l'índex del element del array menor.

El plantejament doncs queda de la següent forma:

- Carregar els valors a l'array
  X[i] , i = [1..N]
  X[i] <- valor integer
- Entrar en un BUCLE repetitiu :
  de i=1 fins i= N-1
  - Assignar a x,k el valor de X[i],i
  - Entrar en un BUCLE repetitiu:
    de j=i+1 fins a N
    - Comparar si X[j] < x
      Si és correcte:
      -assignar a x,K el valor de X[j],j
      per tal de que x,k mantingui sempre
      el valor més petit.
  - Una vegada sortit del segon bucle reassignem valors:
    x[k] <- x[i]
    x[i] <- x
- Finalment acabat el primer bucle treurem per pantalla el resultat.